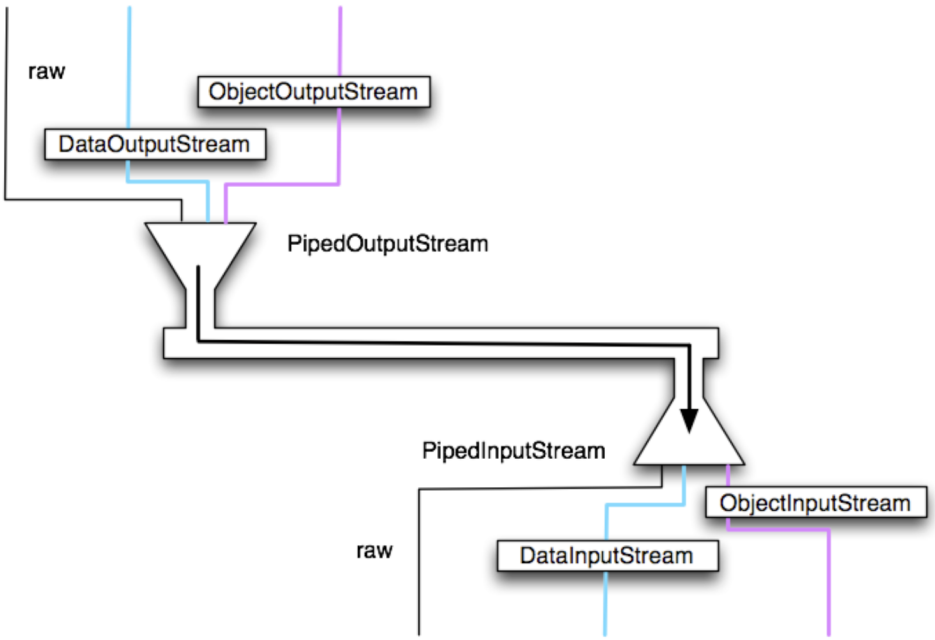
· U C ·

Universidade de Coimbra
Departamento de Engenharia Informática
Sistemas Distribuídos 2018/19

Exercises #2 Socket Programming in Java

Use the files in `StreamsExamples.zip` and `SocketsExamples.zip` to answer the questions.

1. Before we can use sockets, we need to have some means to put data in and take data out of the sockets. For this purpose, we have the Streams. Take a look and run the `StreamDemos.java`, which illustrates the use of raw data, Data Streams and Object Streams. Note that this is done in the simplest of all scenarios without sockets or threads, using the classes `PipedInputStream` and `PipedOutputStream`. The following picture illustrates the flow of data in the program. The "raw data", the Data Stream and the Object Stream all share the same underlying channel. Students should notice that different uses of the stream take place in different moments of time. They cannot be simultaneous, because we have two streams and one "raw" access, but all use the same communication channel.



2. Take a look at the programs `TCPClient+TCPServer`. This is an *echo* service: the server sends back to the client whatever it receives, just changing the string to uppercase. Notice that communication takes place using data streams. The server contains two main classes: `TCPServer` and `Connection`. Note that class `Connection` extends class `Thread`. The reason for this is that whenever a new connection arises (`listenSocket.accept()`) a new thread of class `Connection` is created to handle the new client. Note that a connection is always initiated by the client and that there is one thread for each client in a one-to-one correspondence.

- Play with the application and try to understand in which situations there are Exceptions and what type of Exceptions.

- What is the purpose of the following methods? (Browse the Java documentation to find information.)
  - `setTcpNoDelay()`
  - `setSoTimeout()` / `getSoTimeout()`
  - `setKeepAlive()` / `getKeepAlive()`
  - `setSendBufferSize()` / `getSendBufferSize()`
- Change the code and implement a simple CHAT Service: every message that is sent by one client will be received by all clients. (Hint: this requires a data structure that will be shared by all the Threads at the server and will maintain the reference for each client socket.)

3. Take a look at the programs `UDPClient+UDPServer`. This example makes use of Datagram sockets. In this example, data is written there in raw form. The more important classes in these programs are the `DatagramSocket` and `DatagramPacket`. Note that in `DatagramSocket` the server specifies a fixed port. The client does not specify the port because any port will do. The client lets the operating system pick the port. When it turns to sending data, client and server use the `DatagramPacket`. The client must specify the port where the server is waiting for packets (6789) while the server, in the reply, must find out the original port used by the client:

```
DatagramPacket reply = new DatagramPacket(request.getData(),
                                request.getLength(), request.getAddress(),
                                request.getPort());
```

This line is quite illustrative because it shows how to get a lot of information out of a received datagram, namely the data, the length of the data, the IP address of the sender and the port. The client may be run from the command line as follows: java UDPClient localhost

- What happens when the client or the server finish through a `Ctrl-C`?
- Write some code at the client side that should generate a *timeout* when the client does not receive an answer from the Server within 10 seconds.
- The Server has only *one* UDP socket where it receives messages. Do you think that with only one socket it can receive messages from more than one client? How can the server distinguish the messages from different clients?
- Change the code of the Server so that it works like a Calculator: the client sends two integers and a character (+,-,*,/). The Server receives these parameters, does the math and returns back the result to the client that should be printed out in the client display.

4. Run one `MulticastServer` and two `MulticastClients`. Do the two clients receive the messages sent by the server? Does the server need to know how many clients there are (and their IP addresses)? Run another `MulticastServer`. Do the clients receive the messages sent by the two servers? How would you modify the servers so that they also receive each other's messages?

5. Students should ask themselves how could they use streams with the UDP protocol. The answer lies in the classes `ByteArrayInputStream` and `ByteArrayOutputStream`. Study these classes and make the necessary changes in the baseline code to use streams in the UDP protocol.